Hierarchical Interface-Based Non-blocking Verification

R.J. Leduc[†], B.A. Brandin[‡], and W.M. Wonham[†]

[†]Department of Electrical and Computer Engineering
University of Toronto

[‡]Siemens Corporate Research

email: leduc@control.toronto.edu, bertil.brandin@mchp.siemens.de, wonham@control.toronto.edu

Abstract - *In this paper we present a hierarchical method that breaks up a plant into two subsystems, and restricts the interaction of the subsystems by means of an interface. We present a definition for an interface, and define a set of interface consistency properties that can be used to verify if a discrete-event system (DES) is non-blocking. Each clause of the definition can be verified using only one of the two subsystems; thus the complete system model never needs to be constructed.*

1 Introduction

In the area of Discrete-Event Systems (DES), a common task is to verify that a system (based on a cartesian product of subsystems) is non-blocking. The main obstacle to this task is the combinatorial explosion of the product state space. Although many methods have been developed to deal with this problem (modular control ([1], [9], and [17]), decentralised control ([11] and [14]), model aggregation methods ([2], [3], [6], [8], [16], and [18]), and multi-level hierarchy ([5], [7], [12], [13], [15], and [19])), large-scale systems are still problematic.

In this paper, we present an interface-based hierarchical method to verify if a system is non-blocking. The method splits the given system into two subsystems, and defines an interface DES to regulate how the two subsystems interact. We then present a set of consistency requirements that the interface and subsystems must satisfy. We then state our main result, followed by a small example to illustrate it. We close by discussing application of the method to a large example (750 million reachable states).

2 Introduction to Method

With *hierarchical interface-based non-blocking verification,* what we are proposing is essentially a master-slave system, where a *high level subsystem* sends a command to a *low level subsystem,* which then performs the indicated task and sends back a reply. Figure 1 shows conceptually the structure of the system.

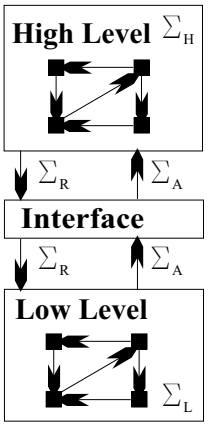

Figure 1: Interface Block Diagram.

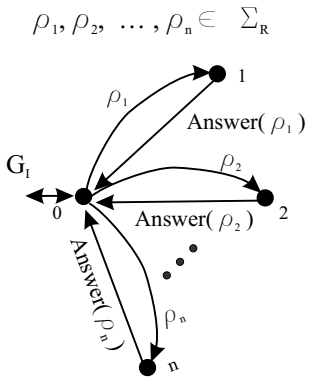

Figure 2: Interface Specification.

To capture the restriction of the flow of information imposed by the interface, the alphabet of the plant (Σ) is split into four disjoint alphabets: Σ_H, Σ_L, Σ_R, and Σ_A. The events in Σ_H are called *high level events* and the events in Σ_L *low level events* as these events appear only in the high level and low level models, respectively.

The alphabets Σ_R and Σ_A are called collectively *interface events.* These events are common to both levels of the hierarchy and represent communication between the two subsystems. More specifically, the events in Σ_R are called *request events* and represent

commands sent from the *high level subsystem* to the *low level subsystem.* Finally, the events in Σ_A are *answer events* and represent the low level's responses to the *request events* (high-level commands). Figure 1 shows conceptually the flow of information in our setting.

3 Interface Definition

To define an *interface,* the designer selects a set of *request events*, and then for each *request event*, the designer defines a set of *answer events.* In essence, the designer defines a map **Answer** : $\Sigma_R \to \mathrm{Pwr}(\Sigma_A)$. For $\rho \in \Sigma_R$, **Answer**(ρ) is the set of possible answers the *low level subplant* could provide after receiving request ρ. For consistency, we add the constraints that the *low level subsystem* must provide at least one response for each request it receives, and that Σ_A does not contain any unused events. Finally, Figure 2 shows how an *interface* is expressed as a DES. The required structure for an *interface* is given by DES G_I.

4 Terminology and Definitions

For our setting, we assume the *high level subsystem* is modelled by DES G_H (defined over event set $\Sigma_H \cup \Sigma_R \cup \Sigma_A$), the *low level subsystem* by DES G_L (defined over event set $\Sigma_L \cup \Sigma_R \cup \Sigma_A$), and the interface by DES G_I. Also, the *high level* will mean $\mathbf{sync}(G_H, G_I)$,[1] and the *low level* $\mathbf{sync}(G_L, G_I)$. The overall structure of the system is displayed in Figure 3.

We next assume that the alphabet partition is specified by $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$ and take the overall system (hereafter referred to as *system*) to be:

$$G = \mathbf{sync}(G_H, G_L, G_I)$$

To simplify the notation in proofs, we introduce some terminology. As we will often be referring to different groupings of events, we define the following subsets:

$\Sigma_I :=$	$\Sigma_R \dot\cup \Sigma_A$	*Interface Events*
$\Sigma_{IH} :=$	$\Sigma_H \dot\cup \Sigma_R \dot\cup \Sigma_A$	*Interface & High Level Events*
$\Sigma_{IL} :=$	$\Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$	*Interface & Low Level Events*

To work with languages defined over subsets of Σ, we define the following natural projections:

$$P_{IH} : \Sigma^* \to \quad \Sigma_{IH}^*$$

[1] The operation **sync** is the synchronous product operation from CTCT [17].

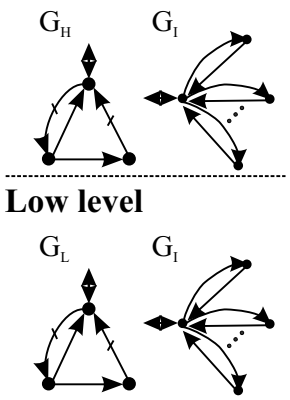

Figure 3: Two Tiered Structure of the System.

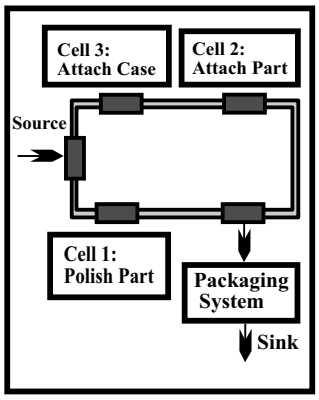

Figure 4: Block Diagram of Plant

$$P_{IL} : \Sigma^* \to \quad \Sigma_{IL}^*$$
$$P_I : \Sigma^* \to \quad \Sigma_I^*$$

and the following useful languages:

$$\mathcal{H} := P_{IH}^{-1}(L(G_H)), \quad \mathcal{H}_m := P_{IH}^{-1}(L_m(G_H)) \quad \subseteq \Sigma^*$$
$$\mathcal{L} := P_{IL}^{-1}(L(G_L)), \quad \mathcal{L}_m := P_{IL}^{-1}(L_m(G_L)) \quad \subseteq \Sigma^*$$
$$\mathcal{I} := P_I^{-1}(L(G_I)), \quad \mathcal{I}_m := P_I^{-1}(L_m(G_I)) \quad \subseteq \Sigma^*$$

We can now represent the language of *system* in terms of its components:

$$\begin{aligned} L(G) :=& \quad L(\mathbf{sync}(G_H, G_L, G_I)) \\ =& \quad P_{IH}^{-1}(L(G_H)) \cap P_{IL}^{-1}(L(G_L)) \cap P_I^{-1}(L(G_I)) \\ =& \quad \mathcal{H} \cap \mathcal{L} \cap \mathcal{I} \end{aligned}$$

Similarly, the marked language of *system* is:

$$L_m(G) = \mathcal{H}_m \cap \mathcal{L}_m \cap \mathcal{I}_m$$

5 Interface Consistency Properties

We now present the interface properties that *the system* must satisfy to ensure that it interacts with the *interface* correctly.

Serial Interface Consistent: The system composed of DES G_H, G_L and G_I, is *serial interface consistent* with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, if the following properties are satisfied:

Multi-level Properties

1. The event set of G_H is Σ_{IH}, and the event set of G_L is Σ_{IL}.
2. G_I is an *interface* for the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$

High Level Properties

3. $(\forall s \in \mathcal{H} \cap \mathcal{I})\ \mathrm{Elig}_{\mathcal{I}}(s) \cap \Sigma_A \subseteq \mathrm{Elig}_{\mathcal{H}}(s)$

Low Level Properties

4. $(\forall s \in \mathcal{L} \cap \mathcal{I})\ \mathrm{Elig}_{\mathcal{I}}(s) \cap \Sigma_R \subseteq \mathrm{Elig}_{\mathcal{L}}(s)$
5. $(\forall s \in \Sigma^*.\Sigma_R \cap \mathcal{L} \cap \mathcal{I})$
   $\mathrm{Elig}_{\mathcal{L}\cap\mathcal{I}}(s\Sigma_L^*) \cap \Sigma_A = \mathrm{Elig}_{\mathcal{I}}(s) \cap \Sigma_A$

   where $\mathrm{Elig}_{\mathcal{L}\cap\mathcal{I}}(s\Sigma_L^*) := \cup_{l\in\Sigma_L^*} \mathrm{Elig}_{\mathcal{L}\cap\mathcal{I}}(sl)$
6. $(\forall s \in \mathcal{L} \cap \mathcal{I})$
   $s \in \mathcal{I}_m \Rightarrow (\exists l \in \Sigma_L^*)\ sl \in \mathcal{L}_m \cap \mathcal{I}_m$

These properties mean the following:

Property 0: The first Property is inherent in the definition of the alphabet partition, $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$. It states that the four alphabets are pairwise disjoint.

Property 1: This property asserts that the high and low levels can only share *request* and *answer events.*

Property 2: This property states that DES G_I is a proper interface of the form defined in Section 3 for the specified alphabet partition.

Property 3: This property asserts that *answer events* must be eligible in the *high level subsystem* (G_H), if the event is eligible in the *interface.*

Property 4: This property asserts that *request events* must be eligible in *the low level subsystem* (G_L), if the event is eligible in the *interface.*

Property 5: This property asserts that after a *request event* has occurred, there exists a path via strings in Σ_L^* to each *answer event* that can follow the *request event.*

Property 6: This property asserts that every string marked by the *interface* and accepted by the *low level subsystem*, can be extended by a low level string to a string marked by the *low level* (both G_I and G_L).

6 Level-wise Non-blocking

In this section, we define the non-blocking requirements each level must satisfy.

Definition: The system composed of DES G_H, G_L, and G_I, is said to be *serial level-wise non-blocking* if the following conditions are satisfied:

(I) $\overline{\mathcal{H}_m \cap \mathcal{I}_m} = \mathcal{H} \cap \mathcal{I}$ *Non-blocking at the high level*

(II) $\overline{\mathcal{L}_m \cap \mathcal{I}_m} = \mathcal{L} \cap \mathcal{I}$ *Non-blocking at the low level*

7 Non-blocking Theorem

We now present our main result, the *serial interface non-blocking theorem.*

Theorem 1 *If the system composed of DES G_H, G_L, and G_I is* serial level-wise non-blocking *and* serial interface consistent *with respect to the alphabet partition $\Sigma := \Sigma_H \dot\cup \Sigma_L \dot\cup \Sigma_R \dot\cup \Sigma_A$, then*

$L(G) = \overline{L}_m(G)$, *where* $G = \mathbf{sync}(G_H, G_L, G_I)$
Proof: *See Leduc [10].*

8 Simple Manufacturing Example

In this section, we present a simple manufacturing example to illustrate the *hierarchical interface-based non-blocking verification* method. The example presented was inspired in part by the examples given in Wang [15], and in Brandin [4]. Table 1 defines abbreviations used for the event labels.

Abbrev.	Meaning	Abbrev.	Meaning
pt	part (item)	str	start
cmpl	complete	attch	attach
fin	finish	ent	enter
rlse	release	lv	leave
pol	polish	recog	recognize
arr	arrive		

Table 1: Abbreviations Used in Event Labels

8.1 Description of Manufacturing Unit

As shown in Figure 4, the manufacturing unit is composed of three cells connected by a conveyor belt. In

front of each cell, is a part acquisition unit that automatically stops a part and holds it until it is given a release command. Parts enter the system at the far left and exit at the bottom right. After the item exits the conveyor system, it goes to a packaging machine.

The associated plant models can be seen in Figure 5, namely **Attach Case to Assembly**, **Polish Part**, **Attach Part to Assembly**, **Packaging System**, and **Path Flow Model**. Of note in the path flow model are the events *recog_A,* and *recog_B*. The acquisition unit in front of cell two is capable of recognising whether a part is of type A or type B.

8.2 Partitioning the System

The first step in the process is to decide which plant models belong to the *high level subsystem,* and which to the *low level subsystem.* The division we have chosen can be seen in Figure 5.

8.3 Augmenting the Low Level Subsystem

We now note that the model for cell two is not well suited to be accessed through an interface. It requires that the decision to attach part A or part B be made after event *take_pt* occurs. To make this functionality available to the upper level, we augment the model by adding the DES **Define New Events** shown in Figure 5. The new request events (*attch_ptA* and *attch_ptB*) will provide the high level with an easy selection method while the new finish events (*finA_attch* and *finB_attch*) will inform the high level of the completion of their respective tasks.

8.4 Defining the Interface

We now have the basic building blocks to create an *interface.* Figure 5 shows the interface DES, G_I. From the diagram, we can see the *request events* and the *answer events.*

We define the alphabet partition $\Sigma := \Sigma_H \dot{\cup} \Sigma_L \dot{\cup} \Sigma_R \dot{\cup} \Sigma_A$ as follows:

Σ_R = {*start_pol*, *attch_ptA*, *attch_ptB*, *start_case*}

Σ_A = {*comp_pol*, *finA_attch*, *finB_attch*, *compl_case*}

Σ_H = {*part_ent*, *part_arr1*, *part_lv1*, *partLvExit*, *str_exit*, *fin_exit*, *part_arr2*, *recog_A*, *recog_B*, *part_lv2*, *part_arr3*, *part_lv3*, *take_item*, *allow_exit*, *package*}

Σ_L = {*take_pt*, *str_ptA*, *str_ptB*, *compl_A*, *compl_B*, *ret_pt*, *dip_acid*, *polish*, *str_rlse*, *attch_case*}

8.5 Designing Low Level Supervisors

Now that we have defined our *interface,* we design the low level supervisors that will provide the functionality for the *request events,* and give meaning to the *answer events.* The idea is for the *low level* to offer well-defined "services" to the *high level.*

We start with cell one. Here we want the sequence *dip_acid-polish* to be repeated twice, after a *start_pol* event occurs. The supervisor is shown in Figure 5, and is labelled **Polishing Sequence**. For cell two, we have to provide supervisors so that the cell reacts appropriately when events *attch_ptA* and *attch_ptB* occur. We also must guarantee that *answer events* *finA_attch* and *finB_attch* only occur when they have the appropriate meaning. The DES **Affix Part** in Figure 5 shows how this is done. Finally, we do nothing for cell three as it is so simple, its functionality being already present.

8.6 Designing High Level Supervisors

Now that the low level functionality is taken care of, we will design high level supervisors that use the *interface.* Figure 5 shows a supervisor (**Sequence Tasks**) that allows a part to visit each cell, executes the appropriate command for the cell and the part type, and then allows the part to leave the conveyor system. The figure also shows a supervisor (**Exit Buffer**) that implements a two item buffer for the packaging system.

8.7 The Final System

With the system components defined, it is time to put them together. Figure 5 shows the complete system. We have labelled which parts correspond to the DES G_H and G_L, the *high* and *low level subsystems.*

We now wish to determine whether the *system* ($G = \mathbf{sync}(G_H, G_L, G_I)$) is non-blocking. For this, we have verified that the system is *serial interface consistent*, and *serial level-wise non-blocking.* We can thus conclude by **Theorem 1** that the system is indeed non-blocking.

8.8 Concurrency of Subsystems

Before concluding this example, we comment on the inherent concurrency of the *high* and *low levels.* Unlike state expansion methods such as Bing [15] and Gohari [7] that expand a high level state into a group of low level states, the interface method is based on the *synchronous product*, limiting information flow, and a set of consistency rules. In general, there is no one-to-one association between a high level state and a set of low level states. This allows the *high level* to remain

active while the *low level* is active, and thus operate concurrently. In the cited state expansion methods, the high level state would remain fixed while the low level becomes active.

This concurrency can be seen in the current example, by noting that once the event *fin_exit* has occurred, the string shown below is then possible.

part_ent part_arrv1 start_pol dip_acid take_item polish

The string clearly shows how the *high level* event *take_item* can occur in the middle of a sequence of *low level* events, thus demonstrating that both levels are active.

9 Car Door Locking Controller

We have also applied our method to a large car door locking example[2] which is composed of 54 DES, and has a composite model of approximately 750 million reachable states. The system was found to be *serial interface consistent*, and to be *serial level-wise non-blocking*,hence non-blocking by **Theorem 1**. The computation was run on an AMD K6 300 Linux workstation with 256MB of RAM, and 1.5GB of swap space. The computation took 19 minutes, and used 91MB of memory. A standard non-blocking verification was also attempted on the monolithic system. It ran for 38 minutes and consumed 1.7GB of memory before running out of memory.

10 Conclusions

Hierarchical interface-based non-blocking verification offers an effective method to model systems with a natural client-server architecture. The method offers an intuitive way to model and design the system. As each requirement can be verified using only one of the two subsystems, the entire plant model never needs to be stored (in computer memory), offering potentially significant savings in computation.

References

[1] N. Alsop. *Formal Techniques for the Procedural Control of Industrial Processes.* PhD thesis, Department of Chemical Engineering and Chemical Technology, Imperial College of Science, Technology and Medicine, London, 1996.

[2]Unfortunately, the details of this example can't be released at this time for proprietary reasons.

[2] Rajeev Alur and Thomas A. Henzinger. Local liveness for compositional modelling of fair reactive systems. In *Proc. of seventh Int. Conf. on Computer-aided Verification, Lecture Notes in Computer Science*, pages 166–179, 1995.

[3] Adnan Aziz, Vigyan Singhal, and Gitanjali M. Swamy. Minimizing interacting finite state machines: A compositional approach to language containment. In *Proc. of IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors*, pages 255–261, Cambridge, Massachusetts, Oct 1994.

[4] Bertil Brandin and François Charbonnier. The supervisory control of the automated manufacturing system of the AIP. In *Proc. Rensselaer's 1994 Fourth International Conference on Computer Integrated Manufacturing and Automation Technology*, pages 319–324, Troy, Oct 1994.

[5] Y. Brave and M. Heymann. Control of discrete event systems modeled as hierarchical state machines. *IEEE Trans. on Automatic Control*, 38(12):1803–1819, Dec 1993.

[6] P.E. Caines and Y.J. Wei. The hierarchical lattices of a finite machine. *Systems Control Letters*, 25:257–263, July 1995.

[7] Peyman Gohari-Moghadam. A linguistic framework for controlled hierarchical DES. Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1998.

[8] Susanne Graf and Bernhard Steffen. Compositional minimization of finite state systems. In *Proc. of the 1990 Workshop on Computer-Aided Verfication*, pages 186–196, June 1990.

[9] Ryan Leduc. PLC implementation of a DES supervisor for a manufacturing testbed: An implementation perspective. Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1996.

[10] Ryan. Leduc. *Interfaced-Based Hierarchical Supervisory Control.* PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, in preparation.

[11] F. Lin and W.M. Wonham. Decentralized control and coordination of discrete-event systems with partial observations. In *Proc. 27th IEEE Conf. Decision Contr.*, pages 1125–1130, Dec 1988.

[12] Hong Liu, Jun-Cheol Park, and Raymond E. Miller. On hybrid synthesis for hierarchical structured petri nets. Technical report, Department of Computer Science, University of Maryland, College Park, MD, 1996.

[13] Chuan Ma. A computational approach to top-down hierarchical supervisory control of DES. Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1999.

[14] Karen Rudie and W.M. Wonham. Think globally, act locally: decentralized supervisory control. *IEEE Trans. on Automatic Control*, 37(11):1692–1708, Nov 1992. Reprinted in F.A. Sadjadi (Ed.), Selected Papers on Sensor and Data Fusion, 1996; ISBN 0-8194-2265-7.

[15] Bing Wang. Top-down design for RW supervisory control theory. Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1995.

[16] K.C. Wong. *Discrete-Event Control Architecture: An Algebraic Approach*. PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1994.

[17] W.M. Wonham. *Notes on Control of Discrete-Event Systems*. Department of Electrical and Computer Engineering, University of Toronto, 1999. Notes and CTCT software can be downloaded at http://odin.control.toronto.edu/DES/.

[18] H. Zhong and W.M. Wonham. On the consistency of hierarchical supervision in discrete-event systems. *IEEE Trans. on Automatic Control*, 35(10):1125–1134, Oct 1990.

[19] Meng Chu Zhou, David T. Wang, and Israel Mayk. Using petri nets for object-oriented design of command and control systems. *International Journal of Intelligent Control and Systems*, 2(2):287–300, 1998.

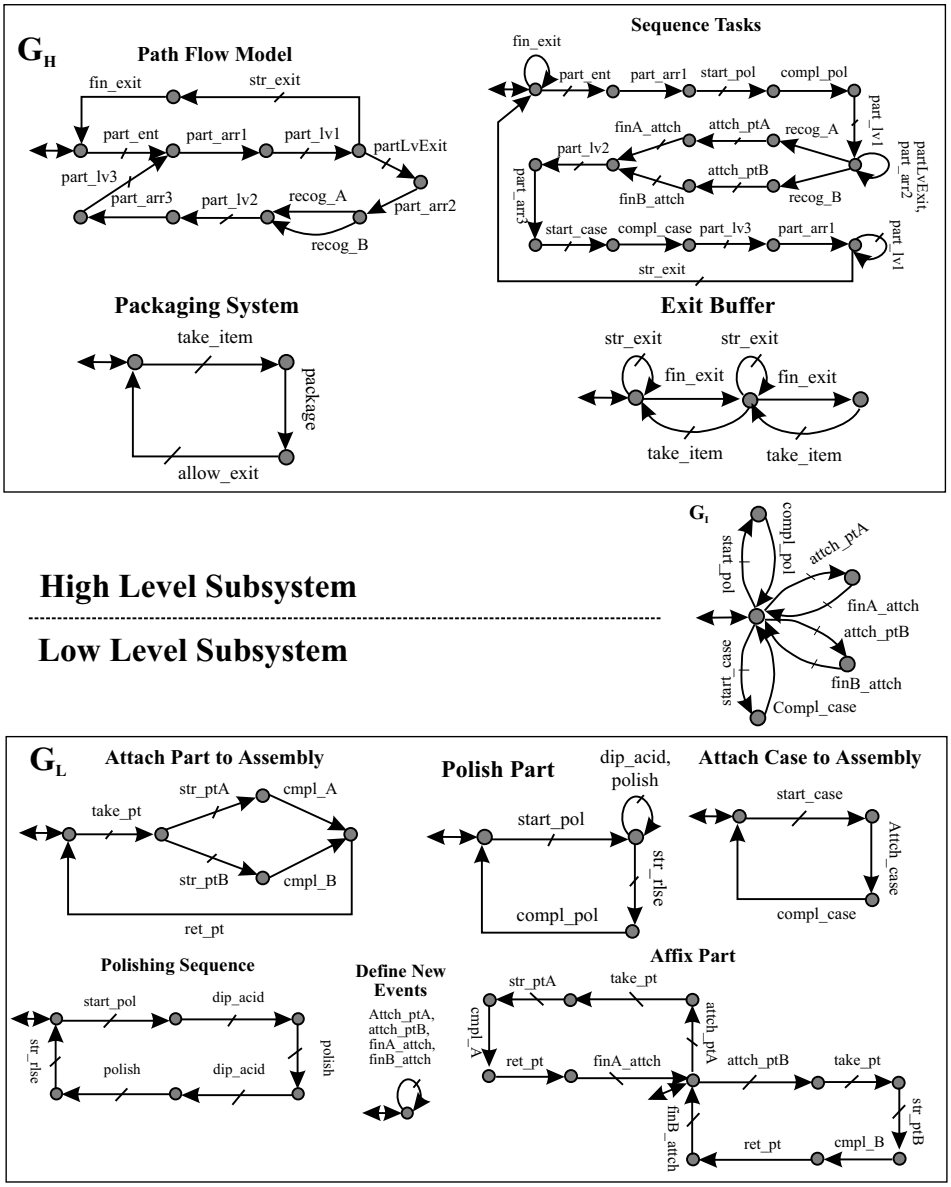

Figure 5: Complete System Definition